Exam1 — ECE 438

April 20^{th}, 2020

Name:

- This is a open-book, open-note exam.
- You *must* work on this exam by yourself!
- Calculators are permitted, but no communication devices of any kind are allowed.
- Each question is marked with its number of points.
- Show your answers in the space provided for them. Write neatly and be well organized.
- Show your work if you want to get credit!

Good luck!

Problem	Maximum	Score
1	10	
2	25	
3	15	
4	20	
5	15	
6	15	
Total	100	

1. **(10 points)** Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3.0 GHz clock rate with a CPI of 1.5. P2 has a 2.8 GHz clock rate with a CPI of 1.2. P3 has a 1.6 GHz clock rate with a CPI of 0.8.

   (a) Which processor has the highest clock rate? Calculate the clock period of each processor.

P_1 has the highest clock rate

P_1 clock period $= 1/3.0\ GHz = 333.3\ ps$

P_2 clock period $= 1/2.8\ GHz = 357.1\ ps$

P_2 clock period $= 1/1.6\ GHz = 625.0\ ps$

   (b) Calculate the Million Instruction Per Second (MIPS) of each processor. Which one has the highest MIPS rating?

P_2 has the highest MIPS rating

P_1 MIPS $= \frac{3.0 \cdot 10^9\ cc/s}{1.5\ cc/instr} \cdot \frac{1}{10^6} = 2.0 \cdot 10^3\ MIPS$

P_2 MIPS $= \frac{2.8 \cdot 10^9\ cc/s}{1.2\ cc/instr} \cdot \frac{1}{10^6} = 2.33 \cdot 10^3\ MIPS$

P_3 MIPS $= \frac{1.6 \cdot 10^9\ cc/s}{0.8\ cc/instr} \cdot \frac{1}{10^6} = 2.0 \cdot 10^3\ MIPS$

   (c) The same benchmark is compiled for each of the processors such that P3 executes 18 million instructions to completion (i.e. has a dynamic instruction count of 20 million instructions), while P1 and P2 have a dynamic instruction count of 22.0 million instructions. Calculate the execution time of each processor? Which processor is fastest?

oops! I will accept either here!

P_1 $t_{execute} = 22 \cdot 10^6\ instr / 2.0 \cdot 10^3\ MIPS = 11\ ms$

P_2 $t_{execute} = 22 \cdot 10^6\ instr / 2.33 \cdot 10^3\ MIPS = 9.44\ ms$

P_3 $t_{execute} = 20 \cdot 10^6\ instr / 2.0 \cdot 10^3\ MIPS = 10\ ms$

P_2 is fastest

   (d) Assuming the reference machine takes 83 milliseconds to execute the given benchmark, calculate the speedup of each processor compared to the reference machine.

This is essentially asking us to calculate a SPEC ratio if the benchmark was SPEC.

speed up of $P_1 = \frac{83ms}{11ms} = 7.55x$

speed up of $P_2 = \frac{83ms}{9.44ms} = 8.79x$

speed up of $P_3 = \frac{83ms}{10ms} = 8.3x$

2. **(25 points)** Imagine you designed a small embedded processor and synthesized it for a 45nm process technology node with a target clock rate of 800MHz. During power analysis, you found that at the target clock rate with a supply voltage of 0.9V, this processor draws 1.4W of dynamic power and 0.1W of static power. Consider the following power and energy trade-offs:

   (a) Assuming a cryptographic operation takes 0.5 seconds to complete on your processor, what is the energy per cryptographic operation at the target clock rate?

$P_{total} = 1.4W + 0.1W = 1.5W$

Energy per op $= 1.5W \times 0.5s = 0.75J/op$

   (b) For certain applications, your processor performs cryptographic operations 4x faster than necessary. If you were to slow the clock down to 200MHz without adjusting the voltage, what would be the overall power draw? What would be the energy per cryptographic operation?

$P_{dynamic} = 1.4W \times \frac{200MHz}{800MHz} = 0.35W$

$P_{total} = 0.35W + 0.1W = \underline{0.45W}$

Energy per op $= 0.45W \times 0.5s \times 4 = \underline{0.9J/op}$

   (c) Assuming you could safely drop the voltage to 0.6V when operating at a 200MHz clock, recalculate the power draw and energy per cryptographic operation. Assume the leakage *current* remains the same.

$P_{dynamic} = 1.4W \times \left(\frac{200MHz}{800MHz}\right) \times \left(\frac{0.6V}{0.9V}\right)^2 = 0.155W$

$P_{static} = 0.1W \times \left(\frac{0.6V}{0.9V}\right) = 0.0667W$

$P_{total} = \underline{0.223W}$

Energy per op $= 0.223W \times 0.5s \times 4 = \underline{0.445J/op}$

(d) The above questions looked at how the power and energy can be changed by varying the clock rate and voltage of a processor. Modern processors change these parameters dynamically, using a technique called Dynamic Voltage Frequency Scaling (DVFS). What are some other techniques for reducing energy? Briefly describe these techniques.

- Clock gating turns the clock off when logic is not being used, reducing dynamic energy
- Power gating turns off power to unused logic saving both dynamic and static energy
- Hardware acceleration maps computationally intense routines into more efficient hardware

(e) The technology node that a particular processor is fabricated with can also affect the energy efficiency. Imagine that you resynthesized your processor at a 130nm process technology node with a 1.5V supply voltage. In order to maintain the same transistor count, you set your target clock rate to 200MHz. Assuming the leakage *current* remains the same and that the capacitance of the design approximately scales linearly with the feature size, calculate the dynamic and static power for your processor at the 130nm node. What is the energy per cryptographic operation at the 130nm node and how does this compare to that of the 45nm node?

$$P_{dynamic} = 1.4W \times \left(\frac{130nm}{45nm}\right) \times \left(\frac{1.5V}{0.9V}\right)^2 \times \left(\frac{200MHz}{800MHz}\right)$$
$$= 2.81W$$

$$P_{static} = 0.1W \times \left(\frac{1.5V}{0.9V}\right) = 0.1667W$$

$$P_{total} = 2.81W + 0.167W = \underline{2.98W}$$

$$\text{Energy per op} = 2.98W \times 0.5s \times 4 = \underline{5.95J/op}$$

Compared to the 45nm node the 130nm design requires $\frac{5.95J/op}{0.75J/op} = \underline{7.93x}$ the energy

Now it gets worse if we compare to the 45nm w/ DVFS $\Rightarrow$ running at the same speed $\frac{5.95J/op}{0.445J/op} = \underline{13.37x}$

3. **(15 points)** A program that you would like to run on a MIPS based microprocessor has a dynamic instruction count of $1.0 * 10^6$ and can be divided up as follows:

Instruction	Rate of Occurence	
beq/bne	10%	
j	2%	18%
jal	3%	
jr	3%	
addu	15%	
sub	6%	
lui	2%	42%
sll	6%	
slt/slti	8%	
and/andi	5%	
lw	22%	
sw	12%	40%
lb	4%	
sb	2%	

(a) Assuming the average CPI for the MIPS machine is 1.6 for loads and stores, 1.2 for control flow instructions, and 1.0 for ALU instructions, what is the overall average CPI for the instruction mix shown above?

CPI = 0.18 × 1.2 cc/instr + 0.42 × 1.0 cc/instr
+ 0.4 × 1.6 cc/instr = 1.276 cc/instr

(b) Assuming the machine has a clock rate of 1.5GHz, calculate the execution time of this program.

$$t_{ex} = \frac{10^6 \text{ instr} \times 1.276 \text{ cc/instr}}{1.5 \cdot 10^9 \text{ cc/s}} = 0.851 \text{ ms}$$

(c) If you design another machine that has a 1.6GHz clock rate with an average CPI of 1.3 for loads and stores, 1.2 for control flow instructions and 0.8 for ALU instructions, how much faster would the new machine be?

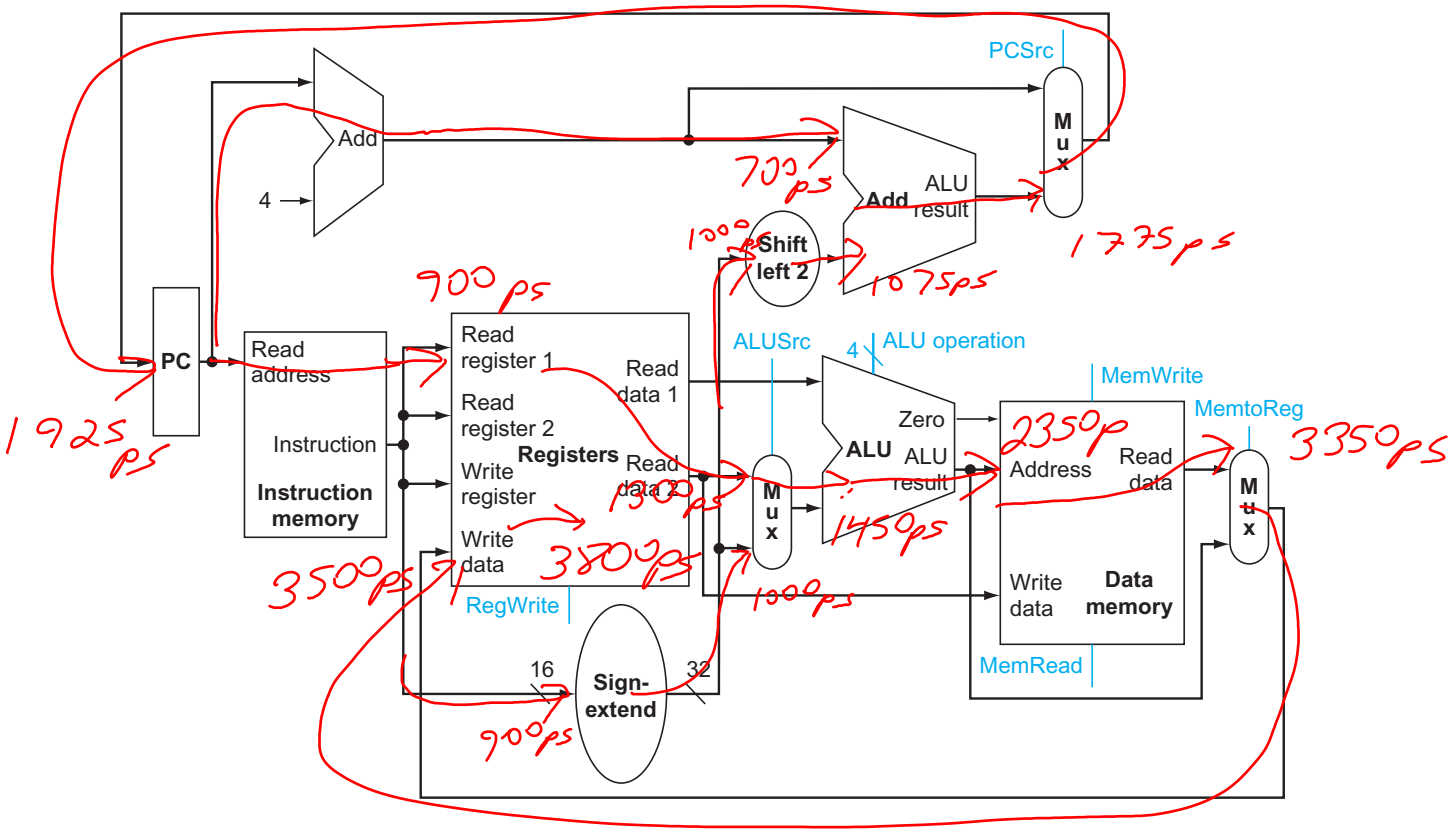
CPI = 0.18 × 1.2 cc/instr + 0.42 × 0.8 cc/instr
+ 0.40 × 1.3 cc/Instr = 1.072 cc/instr

$$t_{ex} = \frac{10^6 \text{ instr} \times 1.072 \text{ cc/instr}}{1.6 \cdot 10^9 \text{ cc/s}} = 0.67 \text{ ms}$$

speedup = 0.851 ms / 0.67 ms = 1.27×

4. **(20 points)** Consider the single-cycle datapath shown below with the following component latencies:

I-mem	Add	Mux	ALU	Reg Read	Reg Write	D-Mem	Sign-Extend	Shift-Left-2
900ps	700ps	150ps	900ps	400ps	300ps	1ns	100ps	75ps



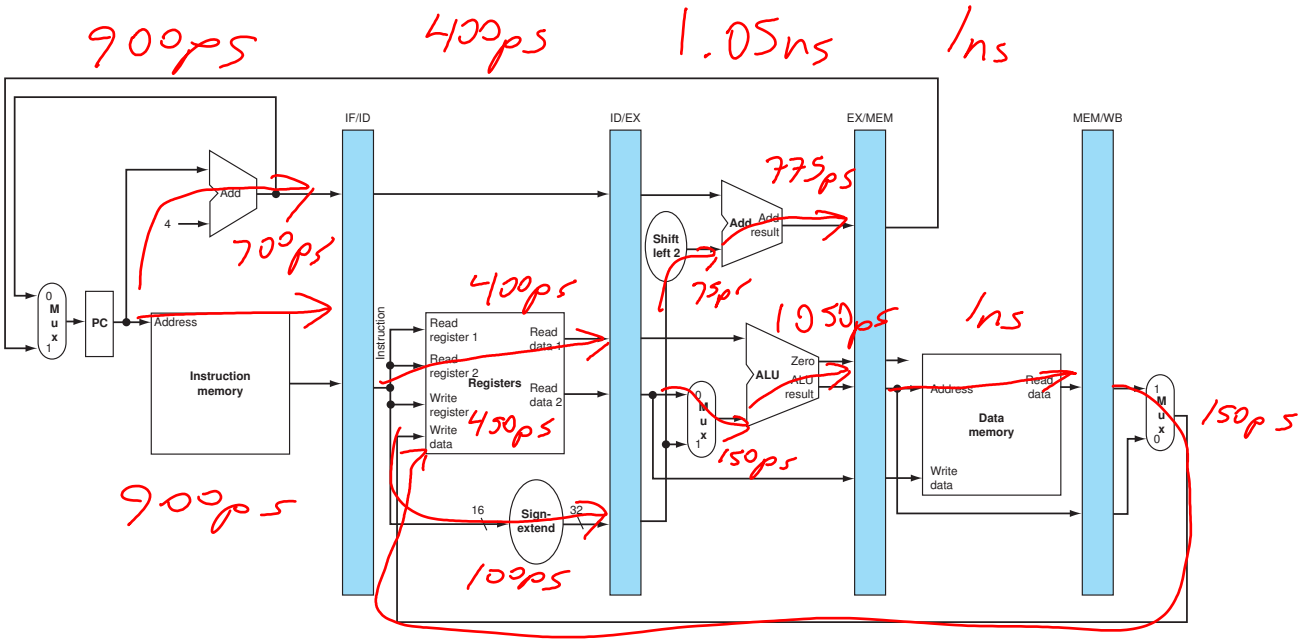
(a) What is the maximum achievable clock rate for the datapath above? Ignore the effects of the control unit.
Hint: Mark the arrival time of valid data on each of the components in the figure above.

$f_{max} = \frac{1}{3800ps} \approx \frac{1}{3.8ns} = 263MHz$

(b) Consider the pipelined datapath shown below. Ignoring the effects of the control unit and flip-flop delays, what is the maximum achievable clock rate?

The longest pipeline stage is actually the EX stage at 1150ps or 1.15ns

Thus $f_{max} = 1/1.050ns \approx 952.4 MHz$



(c) Assuming a long running program with no pipeline hazards, what is the instruction throughput of the above pipelined processor? What is the speedup compared to the single-cycle machine?

The throughput would be 1 instr/cc w/ no hazards, which is same CPI as single cycle.
Speedup = 952.4 MHz / 263 MHz = 3.62x

(d) Briefly describe some nonidealities that erode the performance of pipelining?

Pipeline hazards such as data, structural, and control hazard increase the CPI. Internal fragmentation of the pipeline stages reduces clock rate, as well as clock skew and flip-flop delay.

(e) If you were to add a 3-stage pipelined multiplier to your pipelined processor, such that each stage requires 900ps, calculate the speedup achieved by adding the multiplier assuming the dynamic instruction count is reduced by 72% and the CPI is increased by 11% with the addition of the multiplier.

Because each stage requires 900ps which is less than 1.15ns, the multiplier does not affect our clock rate.

$$\text{Speedup} = \frac{1.11}{0.25} = 4.44x$$

5. **(15 points)** Consider the code below:

```
      add $t2, $t1, 40
Loop:
      lw $t4, 0($t1)
      lw $t5, 40($t1)
      xor $t6, $t4, $t5
      sw $t6, 80($t1)
      addi $t1, $t1, 4
      bne $t1, $t2, Loop
```

(a) Identify the RAW data dependency in the code above.

See above
In addition there are true data dependences between the addi and the lw & sw instructions w/ $t1

(b) Assuming a 5-stage pipelined processor without forwarding or hazard detection, rewrite the code above with `nops` to ensure correct execution. Also assume register write happens in the beginning of the clock cycle, and read happens at the end.

```
      add  $t2, $t1, 40
Loop: lw   $t4, 0($t1)
      lw   $t5, 40($t1)
      nop
      nop
      xor  $t6, $t4, $t5
      nop
      nop
      sw   $t6, 80($t1)
      addi $t1, $t1, 1
      nop
      nop
      bne  $t1, $t2
      nop
```

(c) Assuming a 5-stage pipelined processor with full data forwarding, hazard detection, and support for a single branch delay slot, reorder the code to eliminate as many `nops` as possible and make use of the branch delay slot.

```
      add $t2, $t1, 40
Loop: lw  $t4, 0($t1)
      lw  $t5, 40($t1)
      addi $t1, $t1, 4
      xor $t6, $t4, $t5
      bne $t1, $t2, Loop
      sw  $t6, 76($t1)
```

moved addi between lw and xor to eliminate a stall

moved sw into branch delay slot and adjusted offset because it's now after addi

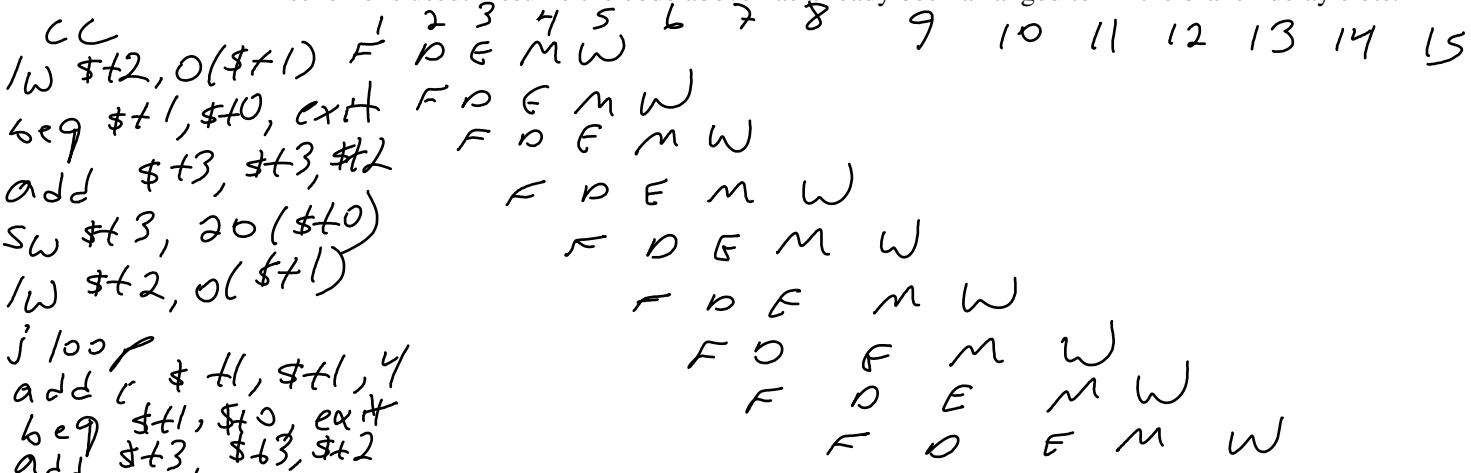
6. **(15 points)** For the following problem, assume a 5-stage pipelined processor with a branch delay slot and branch resolution in the Execute stage. Also assume the pipeline has full forwarding and hardware interlocking. Consider the code below:

```
    lw $t2, 0($t1)
loop:
    beq $t1, $t0, exit #not taken once, then taken
    add $t3, $t3, $t2
    sw $t3, 20 ($t0)
    lw $t2, 0($t1)
    j loop
    addi $t1, $t1, 4
exit:
    sw $t3, 20($t0)
```

notice the code is arranged to avoid stalls!

this one has some weird instruction dependencies because of the branch & jump but this is real code a compiler might produce!

(a) Draw the pipeline execution diagram for the above code when an "assume not taken" branching scheme is used. Assume the code above has already been arranged to fill the branch delay slots.

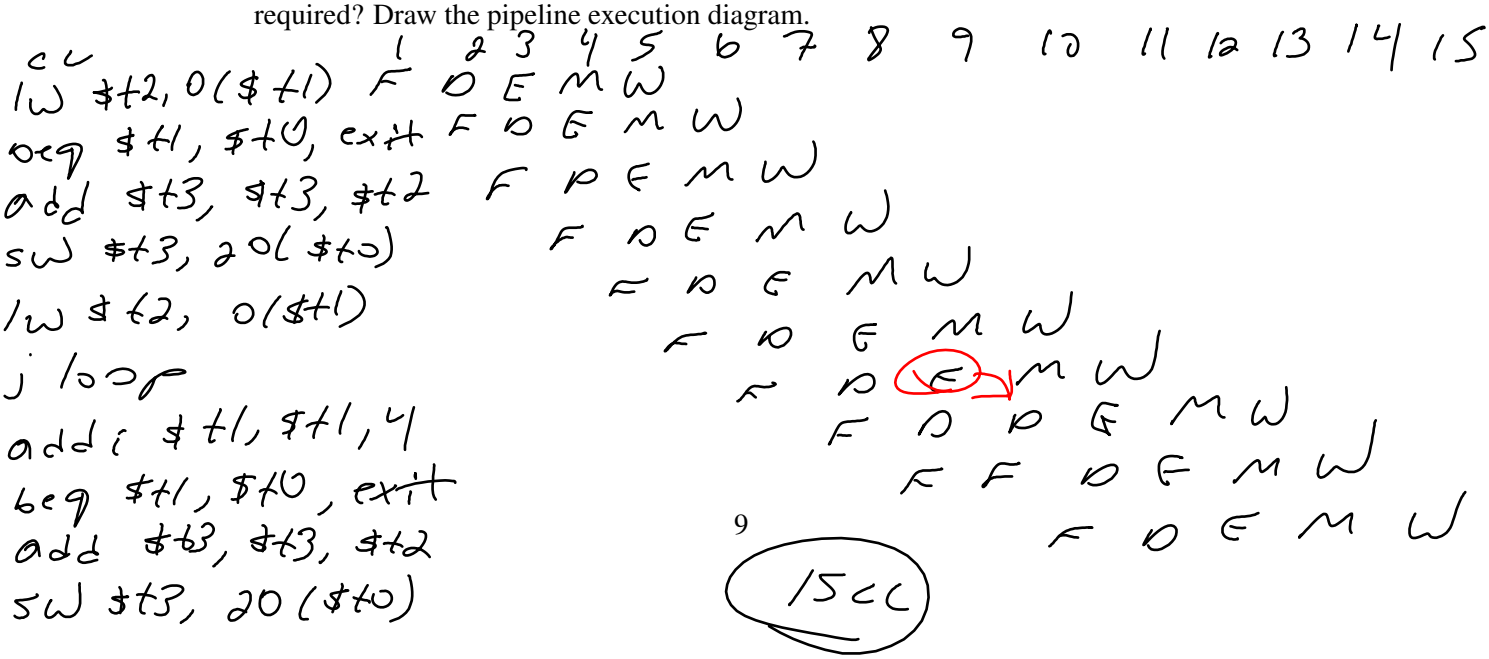


(b) How many clock cycles are required to execute the code above when an "assume not taken" branching scheme is used?

~~4~~ F D E M W

15cc

(c) If the branch resolution was moved to the Decode stage, how many clock cycles would be required? Draw the pipeline execution diagram.



15cc